Stat 274 Pset 3

Karl Jiang

1) Chicago Crime Data

Clean the data first, and make new csv file without all the other data (just keep longitude and latitude)

```
#crimes <- read.csv("C:/Users/KHJ/Downloads/Crimes_-_Map.csv")
#lat_na <- crimes$LATITUDE
#long_na <- crimes$LONGITUDE
#na_i <- is.na(lat_na) | is.na(long_na)
#lat <- lat_na[!na_i]
#long <- long_na[!na_i]
#loc_matrix <- matrix(c(lat, long), ncol = 2)
#colnames(loc_matrix) <- c("LATITUDE", "LONGITUDE")
#loc <- data.frame(loc_matrix)
#write.csv(loc, "crime_data.csv")
crime_loc <- read.csv("crime_data.csv")
```

Single Kernel Density Estimators)

```
#install.packages("sfsmisc")
library(sfsmisc)
#install.packages("caret")
library(caret)
```

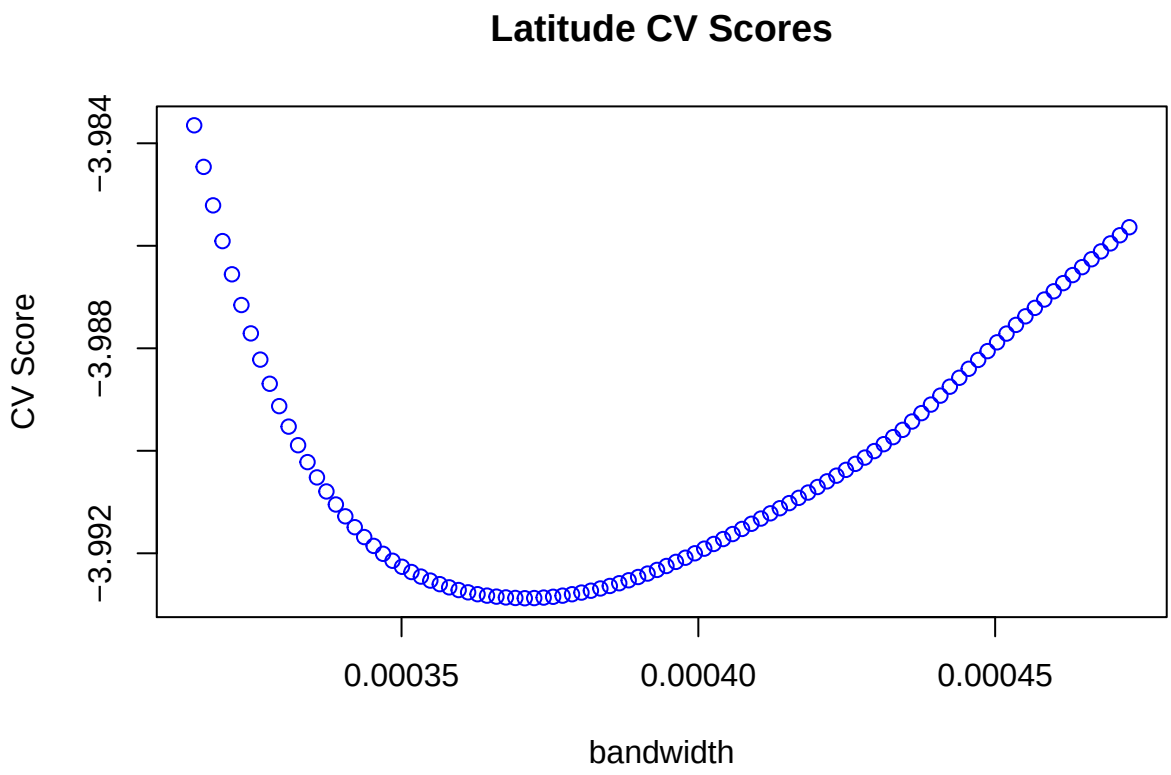
```
## Loading required package: lattice
```

```
## Loading required package: ggplot2
```

```
require(caret)

kfold_dens <- function(data, h, flds) {
  n <- length(data)
  full_dens <- density(data, h)
  dens_integral <- integrate.xy(full_dens$x, full_dens$y^2)

  estimates <- c()
  for(i in 1:length(flds)) {
    k_dens <- density(data[-flds[[i]]], h)
    f_hat_est <- approx(k_dens$x, k_dens$y, xout = data[flds[[i]]])$y
    estimates <- c(estimates, f_hat_est)
  }
  return(dens_integral - 2*mean(estimates))
}

best_h <- function(cv_scores, h) {
  cv_min <- min(cv_scores)
  h_min_index <- match(cv_min, cv_scores)
  h_min <- h[h_min_index]
  return(h_min)
}
```

Latitude)

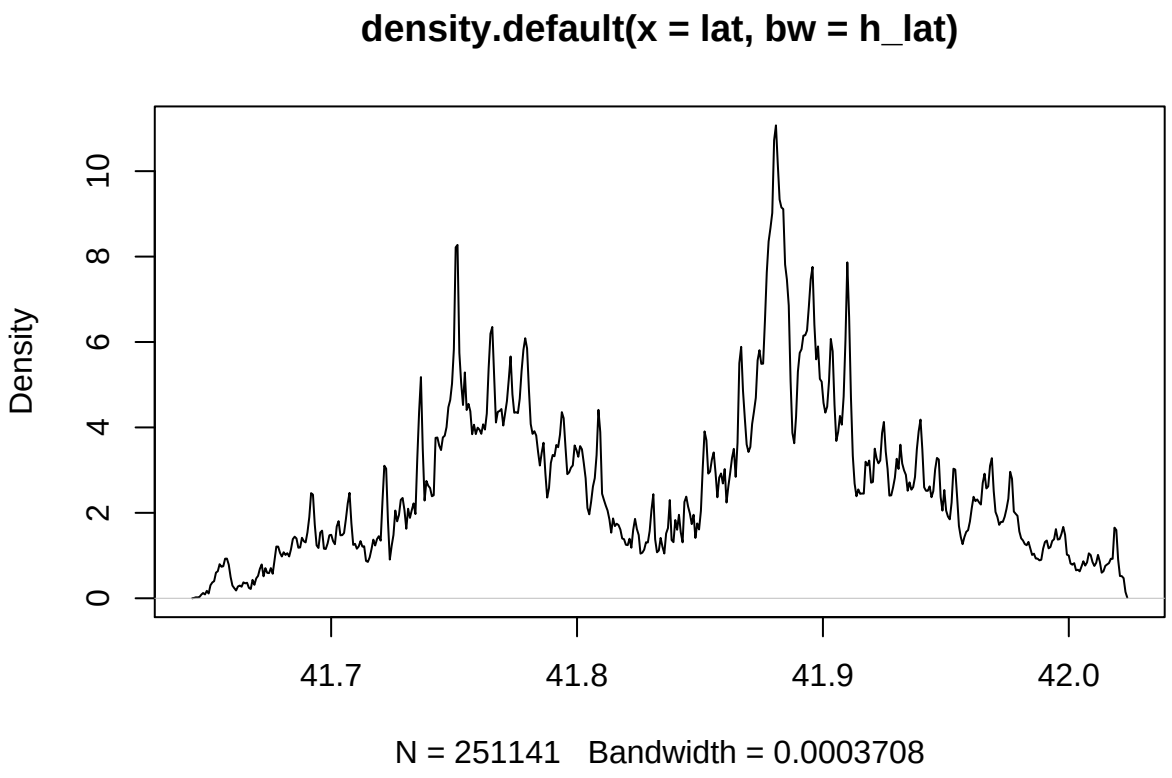
```
lat <- crime_loc$LATITUDE
k <- 5
n <- length(lat)
flds <- createFolds(1:n, k = k, list = TRUE, returnTrain = FALSE)

range_lat <- range(lat)[2] - range(lat)[1]
h <- seq(range_lat / 1200, range_lat / 800, length = 100)

kfold_vals <- c()
for(i in h) {
  kfold_vals <- c(kfold_vals, kfold_dens(lat, i, flds))
}
plot(kfold_vals ~ h, main = "Latitude CV Scores", xlab = "bandwidth", ylab = "CV Score", col = "blue")
```




```
print(h_lat <- best_h(kfold_vals, h))
```

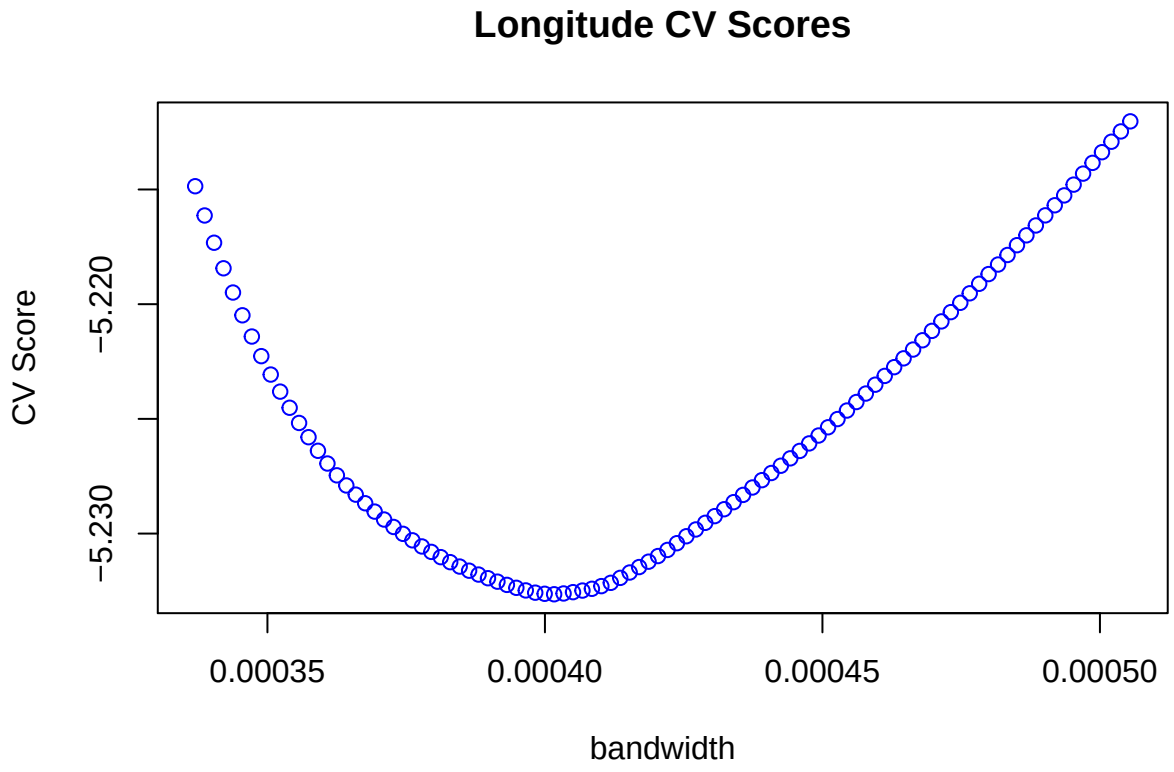
```
## [1] 0.0003707618
```

```
plot(density(lat, bw = h_lat))
```



Longitude)

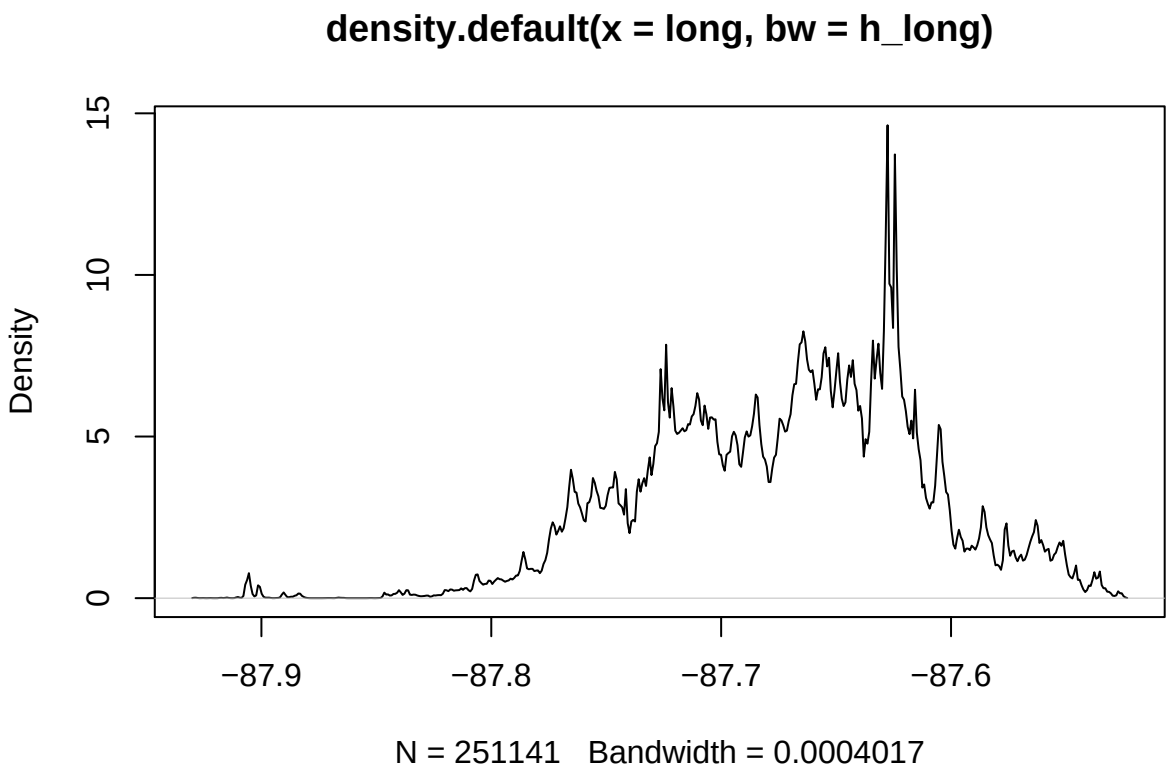
```
long <- crime_loc$LONGITUDE
range_long <- range(long)[2] - range(long)[1]
h <- seq(range_long / 1200, range_long / 800, length = 100)

kfold_vals <- c()
for(i in h) {
  kfold_vals <- c(kfold_vals, kfold_dens(long, i, flds))
}
plot(kfold_vals ~ h, main = "Longitude CV Scores", xlab = "bandwidth", ylab = "CV Score", col = "blue")
```




```
print(h_long <- best_h(kfold_vals, h))
```

```
## [1] 0.000401657
```

```
plot(density(long, bw = h_long))
```



b) 2d Kernel Estimator (Long, Lat)

We use the normal reference rule.

```
h_normal_ref <- function(data, dim = 2){
  Q <- IQR(data)
  s <- sd(data)
  sigma_hat <- min(c(s, Q/1.34))
  return(1.06 * sigma_hat / length(data)^(1/(dim + 4)))
}
#install.packages("MASS")
library(MASS)
library(fields)
```

```
## Loading required package: spam
```

```
## Loading required package: grid
```

```
## Spam version 1.4-0 (2016-08-29) is loaded.
## Type 'help( Spam)' or 'demo( spam)' for a short introduction
## and overview of this package.
## Help for individual functions is also obtained by adding the
## suffix '.spam' to the function name, e.g. 'help( chol.spam)'.
```

```
##
## Attaching package: 'spam'
```

```
## The following objects are masked from 'package:base':
##
##     backsolve, forwardsolve
```

```
## Loading required package: maps
```

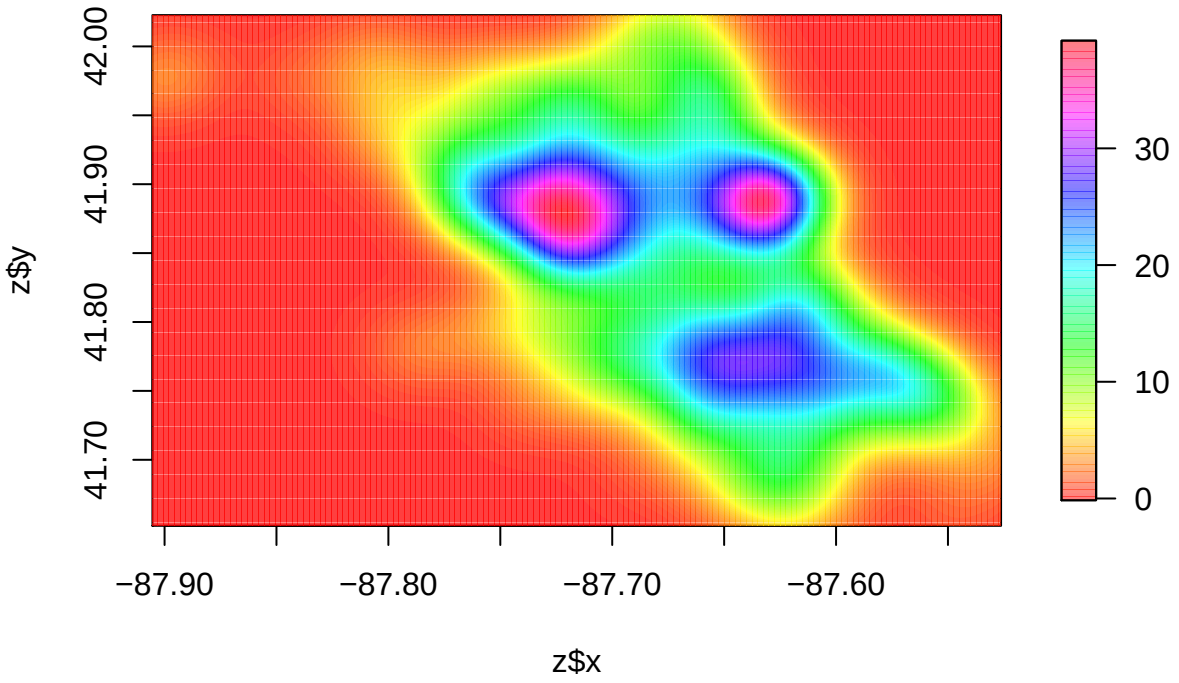
```
print(h_lat <- h_normal_ref(lat))
```

```
## [1] 0.01138443
```

```
print(h_long <- h_normal_ref(long))
```

```
## [1] 0.007875343
```

```
samples <- sample(1:length(lat), size = 1000)
lat_small <- lat[samples]
long_small <- long[samples]
z <- kde2d(long_small, lat_small, n = 1000)

image.plot(z$x, z$y, z$z, col=rainbow(128,alpha=.5))
```



3) Baseball

a)

Let $Y = Z - v$ s.t. $Y_i = Z_i - v_i$, and let $\hat{\theta^y} = \hat{\theta} - v$, where $\hat{\theta}$ is the James Stein estimator with shrinkage towards v (Do not know how to latex the JS, v super script). Then it follows that

$$\hat{\theta^y} = (1 - \frac{(n-2)\sigma^2}{\sum_{i=1}^{n}(Z_i - v_i)^2}(Z - v) = (1 - \frac{(n-2)\sigma^2}{\sum_{i=1}^{n} Y_i^2})(Y)$$

But this is simply the James Stein Estimator with $v = 0$ for $Y_i \sim N(\theta_i^y, \sigma_n^2)$, where $\theta_i^y = \theta_i - v$. We know that the JS estimator achieves lower risk than the MLE (provided that $n >= 3$). Therefore, since the risk of the MLE is $n\sigma_n^2$, $R(\hat{\theta}, \theta) < n\sigma_n^2$ for $n >= 3$.
Note: In the context of this problem, $\sigma_n^2 = \sigma^2$, for clarification.

Best v)

In theory, the best choice of v would be θ, as that would shrink the data towards θ. However, we don't actually know the true θ.

c)

```
m <- 45
b_averages <- c(.400,.346,.378 ,.298,.356,.276,.333,.222,.311,.273,.311,.270,.289,.263,.267,.210,.244,.
b_avgs <- matrix(b_averages, ncol = 2, byrow = TRUE)
y <- b_avgs[,1]
p <- b_avgs[,2]
z <- sqrt(m) * asin(2 * y - 1)
thetas <- sqrt(m) * asin(2 * p - 1)
```

MLE)

As we have shown before, the MLE $\hat{\theta} = Z_i$ is unbiased, with variance σ^2. Therefore, since we have unit variance and $n = 18$ players, the risk/ MSE is just $R(\hat{\theta}, \theta) = 18$ The estimates $\hat{\theta}$ are

```
print(z)
```

```
##  [1] -1.350750 -1.653494 -1.959719 -2.284439 -2.600335 -2.600335 -2.922431
##  [8] -3.251899 -3.605737 -3.605737 -3.954926 -3.954926 -3.954926 -3.954926
## [15] -3.954926 -4.316737 -4.693834 -5.089712
```

JS, v = 0)

```
get_js_coeff <- function(n, sigma, shrink){
  return(1 - (n - 2)*sigma^2 / sum(shrink^2))
}

shrinkage_js <- function(Z, v, sigma){
  n <- length(Z)
  shrink <- Z - v
  coeff <- get_js_coeff(n, sigma, shrink)
  return(v + coeff * shrink)
}
```

As we show in Q4), the risk for any estimator bZ is

$$nb^2 + (b-1)^2 \sum_{i=1}^{n} \theta_i^2$$

Where in the James Stein case

$$b = 1 - \frac{(n-2)\sigma^2}{\sum_{i=1}^{n}(Z_i - v_i)^2}$$

The estimates when v = 0 are:

```
sigma <- 1
v_0 <- rep(0, length(z))
theta_js_zero <- shrinkage_js(z, v_0, sigma)
print(theta_js_zero)
```

```
##  [1] -1.251173 -1.531599 -1.815249 -2.116031 -2.408639 -2.408639 -2.706990
##  [8] -3.012170 -3.339923 -3.339923 -3.663370 -3.663370 -3.663370 -3.663370
## [15] -3.663370 -3.998508 -4.347805 -4.714500
```

Now for the risk.

```
calc_risk_b <- function(b, thetas, sigma_n, v){
  norm_thetas <- sum((thetas - v)^2)
  n <- length(thetas)
  return((1 - b)^2 * norm_thetas + n*b^2*sigma_n^2)
}
b_v_0 <- get_js_coeff(n = length(z), sigma = sigma, shrink = z - v_0)
print(risk_js_zero <- calc_risk_b(b_v_0, thetas, sigma, v_0))
```

```
## [1] 16.53582
```

So the risk when v = 0 is 16.53582.

JS, v = $\bar{Z}$)

The same logic follows from $v = 0$. The estimates are

```
v <- rep(mean(z), length(z))
theta_js_avg <- shrinkage_js(z, v, sigma)
print(theta_js_avg)
```

```
##  [1] -3.009505 -3.056877 -3.104795 -3.155606 -3.205036 -3.205036 -3.255437
##  [8] -3.306991 -3.362358 -3.362358 -3.416998 -3.416998 -3.416998 -3.416998
## [15] -3.416998 -3.473613 -3.532620 -3.594566
```

And the risk is

```
b <- get_js_coeff(n = length(z), sigma = sigma, shrink = z - v)
print(risk_js_avg <- calc_risk_b(b, thetas, sigma, v))
```

```
## [1] 4.454426
```

Quick Comparison)

Shrinking to toward $\bar{Z}$ is better than shrinking to 0 (lower risk 4.5 vs 16.5). This makes sense - after all, the transformed batting averages are all negative, so no point in transforming to 0. The MLE is the worst as per usual since the James Stein estimator make the MLE inadmissable.

d) Foreign Cars

Adding it in the baseball averages will yield a higher risk. As per the 1977 Stein's Paradox article, the risk will only be reduced if the true proportion of imported cars lies in the same range of the estimated batting averages. Otherwise, the risk of both of these problems is increased as the true proportion gets further and further away from the estimated batting averages.

4) James - Stein Estimator

First, note that the risk of the MLE is unbiased, since $\hat{\theta}_i = Z_i$:

$$\mathbb{E}[\mathbb{E}[\hat{\theta}_i] - \theta_i] = \mathbb{E}[\mathbb{E}[Z_i] - \theta_i] = \mathbb{E}[\theta_i - \theta_i] = 0$$

The variance:

$$\mathbb{E}[\hat{\theta}_i - \mathbb{E}[\hat{\theta}_i]]^2 = \mathbb{E}[Z_i - \theta_i]^2 = \sigma_n^2 = 1$$

Therefore, the the total risk is

$$\sum_{i=1}^{n} var(\hat{\theta}_i) + bias(\hat{\theta}_i)^2 = \sum_{i=1}^{n} \sigma_n^2 = 1000$$

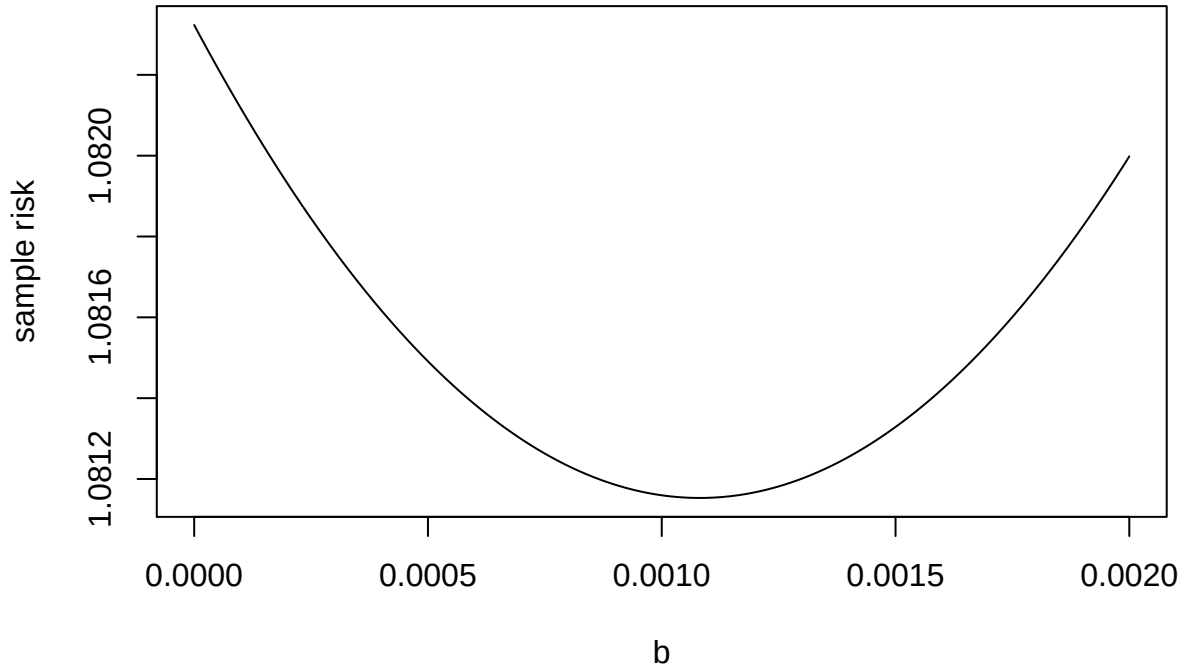
For the MLE since $n = 1000$.

Now for the risk of $\tilde{\theta}$:

$$R(\tilde{\theta}) = \sum_{i=1}^{n} var(\tilde{\theta}_i) + bias(\tilde{\theta}_i)^2 = \sum_{i=1}^{n} var(b*Z_i) + \mathbb{E}[\mathbb{E}[b*Z_i] - \theta_i]^2 = \sum_{i=1}^{n} b^2\sigma_n^2 + \mathbb{E}[b\theta_i - \theta_i]^2 = nb^2 + (b-1)^2 \sum_{i=1}^{n} \theta_i^2$$

Plot risk as function of b:

```r
risk_tilde <- function(b, thetas){
  first <- n * b^2
  second <- (b-1)^2 * sum(thetas^2)
  return(first + second)
}

n <- 1000
thetas <- c()
for(i in 1:n){
  theta <- 1/i^2
  thetas <- c(thetas, theta)
}

b_list <- seq(0, 0.002, length = 1000)
risks <- c()
for(b in b_list){
  risks <- c(risks, risk_tilde(b, thetas))
}
plot(risks ~ b_list, xlab = "b", ylab = "sample risk", type = "l")
```



The optimal value b_* is:

$$\frac{\partial R(\tilde{\theta}, \theta)}{\partial b} = 2b_* n + 2(b-1)\sum_{i=1}^{n} \theta_i^2 = 0$$

$$\implies b_*(n + \sum_{i=1}^{n} \theta_i^2) = \sum_{i=1}^{n} \theta_i^2 \implies b_* = \frac{\sum_{i=1}^{n} \theta_i^2}{n + \sum_{i=1}^{n} \theta_i^2}$$

computing gives us:

```r
norm_thetas <- sum(thetas^2)
print(b_star <- norm_thetas / (norm_thetas + n))
```

```
## [1] 0.001081153
```

So $b_* = 0.001081153$

simulations)

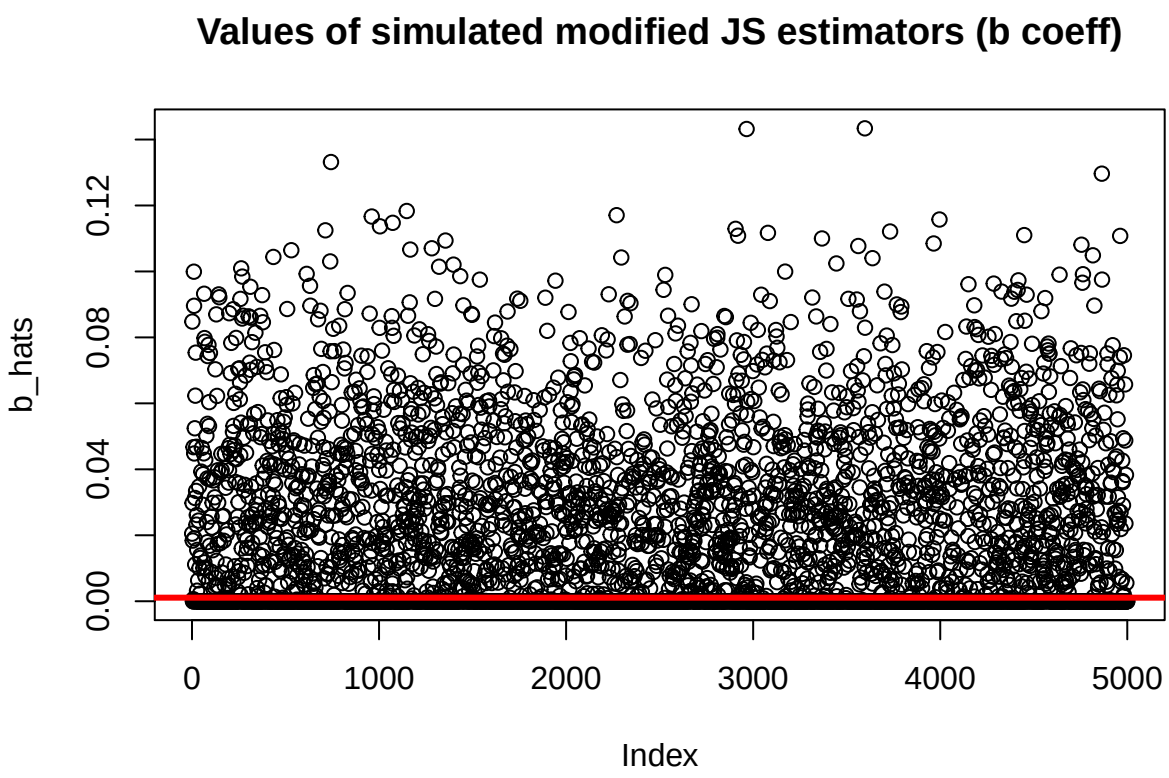
```
#z is vector of data points
get_jse <- function(z, sigma_n = 1){
  n <- length(z)
  val <- 1 - n * sigma_n^2/sum(z^2)
  return(max(val, 0))
}

n_sim <- 5000
b_hats <- c()
for(i in 1:n_sim){
  z <- rnorm(n, mean = thetas)
  b_hats <- c(b_hats, get_jse(z))
}
mean(b_hats)
```

```
## [1] 0.01744984
```

```
median(b_hats)
```

```
## [1] 0.0001864433
```

```
plot(b_hats, main = "Values of simulated modified JS estimators (b coeff)")
abline(h=b_star, col="red", lwd=3)
```



For the most part, the values are are close to 0, just like b_*.

Risk comparison)

From before, the risk of the MLE is $n = 1000$.

```
js_mod_risks <- c()
for(i in 1:n_sim) {
  r <- calc_risk_b(b = b_hats[i], thetas = thetas, sigma_n = 1, v = rep(0, n))
  js_mod_risks <- c(js_mod_risks, r)
}
print(mean(js_mod_risks)) #mean of simulations
```

```
## [1] 1.970411
```

```
print(median(js_mod_risks)) #median of simulations
```

```
## [1] 1.082323
```

And the risk of the mean and median of James Steins Estimators (via simulation) are printed above. Both are significantly smaller than the MLE risk of 1000.

The Pinsker can also be expressed in terms of σ_n^2:

$$\frac{n\sigma_n^2 c^2}{n\sigma_n^2 + c^2}$$

where $c^2 = \sum_{i=1}^{n} \theta_i^2 = \sum_{i=1}^{n} \frac{1}{i^2}$

```
print(p_bound <- (1000 * norm_thetas)/(1000 + norm_thetas))
```

```
## [1] 1.081153
```

The modified JS estimator nearly achieves the risk of the pinsker bound, 1.081153. The MLE is very far, not even close. That the modified (mean and median) JS estimator is close to Pinsker bound, which makes sense since the James Stein estimator is asymptotically minimax.

5) Normal Means Estimation

I use a modification of the James Stein Estimator as per Remark 7.45 in Wasserman. I use the modified version since "taking the positive part of the shrinkage factor cannot increase the risk." (Wasserman) Finally, because the plot (see below) appears to have values strictly greater than 0, I shrink toward the mean of the data (This was the same setting as in question 3, so why not try it here?). So instead of using Z as they did in 7.45, I use $Y = Z - \bar{Z}$ and $\hat{\theta^y} = \hat{\theta} - \bar{Z}$

The plot below shows the Modified (blue) and Regular (red) James Steins Estimators with shrinkage $v = \bar{Z}$:

```
#returns estimates of shrinkage modifified JS estimator
jse_v <- function(z, sigma_n, v){
  n <- length(z)
  y <- z - v
  val <- 1 - n * sigma_n^2/sum(y^2)
  theta_hat <- v + max(val, 0) * y
  return(theta_hat)
}

assn3prob5data <- read.table("assn3prob5data.txt", quote="\"", comment.char="")
Z <- assn3prob5data$V1
Z_sd <- mad(Z)
v <- rep(mean(Z), length(Z))
```

```r
#James Stein
js_estimates <- shrinkage_js(Z = Z, v = v, sigma = Z_sd^2)
js_estimates_mod <- jse_v(z = Z, sigma_n = Z_sd, v)

write(js_estimates_mod, file = "assn3-prob5-karljiang.txt")

jss <- sort(js_estimates)
jss_mod <- sort(js_estimates_mod)
Z_sorted <- sort(Z)
x <-1:length(Z_sorted)
plot(Z_sorted ~ x, main = "Sorted data", xlab = "sorted indices", ylab = "values")
lines(jss, col = "red", lwd = 3)
lines(jss_mod, col = "blue", lwd = 3)
```